Curriculum "Software: Programming, Principles, and Technologies"

Objectives/visions – *The main theme of this track is the design and development of reliable, secure, scalable, usable, and efficient software. Students will improve their scientific and technological knowledge of software tools, also developing competitive skills for job interviews in major ICT companies and for applications to PhD programs (both at the national and international level). The track will technically address the scientific and engineering challenges arising from digital systems and applications, critical to modern society, with a glimpse of key infrastructures in widespread use today, ranging from multiprocessors, embedded devices and mobile phones through to networking systems, internet services and smart things.*

Software is at the heart of this track from both theoretical and practical viewpoints. The driving idea is that formal techniques and automated proof assistants are gaining momentum in industrial software design, analogously to what happened for type systems and syntactic interfaces. The design and the deployment of applications over the aforementioned infrastructures are their natural experimental scenario, also addressing new emerging computing platforms.

Key ingredients in the track will be the foundations and the practice of programming for complex software building and the efficient design of algorithms to solve challenging problems, relying on guaranteed properties for correctness and performance. The learning core will cover the frontiers of algorithms, formal specifications and models, programming practices, software engineering, compiling and analysis techniques. Elective courses will be chosen by the students among application-oriented topics and more specialized subjects. Training will be enriched by laboratories of hands-on activities on themes that are currently challenging in academia and industry. The emphasis will be on methods that are applicable in practice as well as on their underlying supporting theories. The end goal is to convey all the above learning activities to help students to conceive innovative software solutions.

Studies plan

	Course name	CFU
	48 CFU OF:	
1	Competitive programming and contests	6
2	Languages, compilers and interpreters	9
3	Principles for software composition	9
4	Algorithm design	9
5	Software validation and verification	9
6	Laboratory for innovative software	6
7-11	**39 CFU (3 da 9 e 2 da 6) OF:**	
	Advanced programming (WTW)	9
	Smart applications (AI)	9
	Advanced software engineering (ICT)	9
	Distributed systems: paradigms and models (WTW)	9
	Information retrieval (KD)	6
	Machine learning (AI)	9
	Bioinformatics (KD)	6
	Foundations of computing	6
	Security methods and verification (WTW)	6
	ICT infrastructures (ICT)	6
	Computational mathematics for learning and data analysis (KD-AI)	9
	Mobile and cyberphysical systems (ICT)	9
	33 CFU OF:	
12	Free choice	9
	Thesis	24

Syllabus

Competitive programming and contests (Lab) [6 CFU]

The goal of the course is to improve programming and problem solving skills of the students by facing them with difficult problems and by presenting the techniques that help their reasoning in the implementation of correct and efficient solutions. The importance of these skills has been recognized by the most important software companies worldwide, which evaluate candidates in their job interviews mostly by the ability in addressing such difficult problems. A natural goal is to involve the students in the intellectual pleasure of programming and problem solving, also preparing them for the most important international online contests, such as TopCoder, HackerRank, CodeChef, Facebook Hacker Cup, Google Code Jam and so on, for internships in most important companies and their interviews. A desirable side-effect of the course could be to organize and prepare teams of students for the ACM International Collegiate Programming Contests. The course will give the opportunity to uniform students' background in algorithms and programming in view of the subsequent courses.

- *An official language for contests: C++ and its standard template library*
- *Efficient code: programming, benchmarking and profiling*
- *Real-world applications of sorting*
- *Basic data structures: priority queues, search trees, and hash maps*
- *Advanced data structures: union-find, Fenwick tree, interval trees, range-minima query*
- *Basic string algorithms*
- *Basic graph algorithms*
- *Fast optimization with dynamic programming*
- *Computational geometry*

Each topic of the above syllabus will be covered by

- *offering a quick recap of the related concepts from an introductory class on algorithms;*
- *programming and engineering fast software solutions for real-life computational problems;*
- *learning how to recognize their applicability through contests and experimentation.*

Languages, compilers and interpreters [9 CFU]

The course teaches the core of compilation, program analysis techniques used in compilers, and software development tools to improve productivity and reliability. Emphasis is placed on the methodology of applying formal abstractions to writing complex software, using compilers as an example. The course will explore the basic static techniques that are the cornerstone of a variety of program analysis tools, including optimizing compilers, just-in-time compilers, program verifiers, bug finders and code refactoring tools. As case studies, tools developed within the LLVM Compiler Infrastructure will be analyzed and used in experimentations.

Compilation phases and their formal foundations [5 CFU]

- *Front-end: lexical analysis, parsing, syntax-directed translation [2 CFU]*
- *The LLVM compiler infrastructure: architecture, tools, and LLVM intermediate code [1 CFU]*
  - *Generating LLVM bytecode from typical programming constructs*
- *Machine independent optimization: Control flow graph, data-flow analysis and examples of data-flow based optimization techniques [1 CFU]*
- *Pointer analysis and Register Allocation [0.5 CFU]*
- *Just-in-time compilation [0.5 CFU]*

Innovative functional aspects of programming languages [2 CFU]

- *Haskell: interpreter, type system, type inference*
  - *Handling overloading with type classes*
  - *Side effects and Monads*
  - *Compiling Haskell: supercombinators*
- *Lambda expressions and stream API in Java 8 and in other languages*

Generative Programming [2 CFU]

- *MetaProgramming and Reflection*
- *Runtime code generation*
- *Partial evaluation*

Principles for software composition [9 CFU]

This course introduces concepts and techniques in the study of advanced programming languages, as well as their formal logical underpinnings. The central theme is the view of individual programs and whole languages as mathematical entities about which precise claims may be made and proved. The course will cover the basic techniques for assigning meaning to programs with higher-order, concurrent and probabilistic features (e.g., domain theory, logical systems, well-founded induction, structural recursion, labelled transition systems, Markov chains, probabilistic reactive systems) and for proving their fundamental properties, such as termination, normalisation, determinacy, behavioural equivalence and logical equivalence. In particular, some emphasis will be posed on modularity and compositionality, in the sense of guaranteeing some property of the whole by proving simpler properties of its parts. Emphasis will be placed on the experimentation of the introduced concepts with state-of-the-art tools.

- *Introduction and background [1 CFU]*
- *Induction and recursion, partial orders, fixed points, lambda-notation [1 CFU]*
- *Functional programming with Haskell and analysis of higher-order functional languages [1 CFU theory and 1 CFU exercises and experimentation]*
- *Concurrent programming with Google Go and Erlang and analysis of concurrent and non-deterministic systems [2 CFU theory and 1 CFU exercises and experimentation]*
- *Code orchestration with Orc and analysis of coordination languages [1 CFU theory and experimentation]*
- *Models and analysis of probabilistic and stochastic systems [1 CFU theory and experimentation]*

Algorithm design [9 CFU]

The course focuses on developing algorithmic design skills, exposing the students to complex problems that cannot be directly handled by standard libraries (being aware that several basic algorithms and data structures are already covered by the libraries of modern programming languages), thus requiring a significant effort in problem solving. These problems involve all basic data types, such as integers, strings, (geometric) points, trees and graphs, as a starting point and the syllabus is structured to highlight the applicative situations in which the corresponding algorithms can be successfully applied. Brainstorming activities will be central to help students learning from their mistakes. The level of detail in each argument can be adapted year-by-year to some trending topics, and will be decided according to requests coming from other courses and/or specific issues arising in, possibly novel, applicative scenarios.

- *Exploring the algorithms behind standard libraries [2 CFU]*
- *External-memory and cache-efficient algorithms [2 CFU]*
- *Randomized algorithms [2 CFU]*
- *Approximation algorithms and complexity [2 CFU]*
- *Argument chosen from an emerging scenario [1 CFU]*

Software validation and verification [9 CFU]

The goal of the course is to introduce techniques for verifying and validating software properties, either by analysing a model extracted from a program with model checking, or by testing the software before (the next) deployment, or equipping the running software with tools that monitor its execution.

- *Specifying software properties [2 CFU]*
    - *Assertions*
    - *Invariants, Safety and Liveness Properties, Fairness*
    - *Temporal logics: LTL, CTL, CTL**
- *Model Checking [4 CFU]*
    - *Transition systems and Program graphs*

- *Checking regular safety properties*
- *Checking omega regular properties with Büchi automata*
- *Overview of Promela-SPIN and SMV*
- *Extracting models from Java source code: BANDERA*

- *Testing [3 CFU]*
  - *Coverage criteria and metrics: statement, function, branch, path and data-flow coverage*
  - *Test cases selection, prioritization and minimization*
  - *Automatic generation of test cases*

  Topics to be chosen among:
  - *Component-based and Service-oriented system testing*
  - *Object-oriented testing and Junit*
  - *Access Control Systems testing*
  - *Performance and other non-functional aspects testing*

Laboratory for innovative software [6 CFU]

Practical development of software requires an understanding of successful methods for bridging the gap between a problem to be solved and a working reliable software system. This course will train the student to develop large software systems working in real projects by exploiting the techniques and the skills acquired in the fundamental courses of the curriculum.

The main novelty of the course is the attempt to balance traditional lectures and experimental activities with technical meetings with software architects of innovative software enterprises. During the course students will face and deal with the up-to-date issues of software design, implementation and testing of real projects. In this way students will also learn how to inspect actively software solutions.

Each time the course is offered the design and implementation of a new innovative software artifact will be addressed, however the main underlying theme will always be building reliable code. To this purpose the course experiments modern techniques for making software more robust. These techniques include, but are not limited to:

- Ad hoc static code analyses and tools.
- Model checkers.
- Code verification.
- Machine learning techniques applied to code analysis.
- Undefined behavior detectors.
- Testing frameworks.
- Language-based security frameworks.

By the end of this class, students will have developed skills in three distinct competency areas

Reliable coding:

Writing code that is well organized at a high level; exploiting the best programming language features appropriately and avoiding troublesome ones; applying sophisticated idioms to structure code elegantly; using innovative toolkits to check program properties including automatable unit tests in the code base; preventing security attacks.

Design:

Analyzing problems to understand what the tricky aspects are; identifying key design issues, and analyzing their tradeoffs; selecting features for a minimal viable product.

Professionalism:

Constructing and delivering presentation of the deployed software; collaborating with team members; making constructive critiques.

Foundations of computing [6 CFU]

The goal of the course is to present the mathematical foundations of some advanced models of computation. The contents can vary along the years. In the current instance, detailed below, the focus is on algebraic and categorical foundations of calculi for higher-order and concurrent computing. Future instances of the course will focus on the mathematical foundations of other computational models, like

Quantum Computing and Biologically Inspired Computational Mechanisms. No prerequisites are required except for some elementary knowledge of logic and algebra.

- *Simply typed lambda calculus*
- *Curry-Howard isomorphism*
- *PCF and its cpo model, with applications to functional programming languages*
- *Elements of recursive and polymorphic types, with applications to object oriented programming languages*
- *Categories as partial algebras*
- *Monoidal, cartesian and cartesian closed (CCC) categories*
- *CCC as models of simply typed lambda calculus*
- *Algebraic specifications, categories of models and adjunctions*
- *Petri nets and their (strictly) symmetric monoidal models*
- *Labelled Transition Systems (LTS) as coalgebras*
- *The Calculus for Communicating Processes (CCS) and its bialgebraic models*
- *The Pi-Calculus and its presheaf coalgebraic models*

Computational modelling and simulation [6 CFU] (non attivato)

The objective of this course is to train experts in systems modelling and analysis methodologies. Of course, this will require understanding, to some degree of detail, the mathematical and computational techniques involved. However, this will be done with the aim of shaping good modellers, that know the advantages/disadvantages/risks of the different modelling and analysis methodologies, that are aware of what happens under the hood of a modelling and analysis tool, and that can develop their own tools if needed.

The course will focus on advanced modelling approaches that combine different paradigms and analysis techniques: from ODEs to stochastic models, from simulation to model checking. Case studies from population dynamics, biochemistry, epidemiology, economy and social sciences will be analysed. Moreover, synergistic approaches that combine computational modelling techniques with data-driven methodologies will be outlined.

- *Modelling with ODEs: examples*
- *(Timed and) Hybrid Automata: definition and simulation techniques*
- *Stochastic simulation methods (Gillespie's algorithm and its variants)*
- *Hybrid simulation methods (stochastic/ODEs)*
- *Rule-based modelling*
- *Probabilistic/stochastic model checking: principles, applicability and tools*
- *Statistical model checking*
- *Process mining (basic notions)*

Security methods and verification [6 CFU] (vedi WTW)